Rechnernetze - Belegdokumentation

Jonathan Vincent Cremer

January 20, 2019

1 Aufgabenstellung

Ziel dieser Belegarbeit war es "ein Programm (client + server) zur Übertragung beliebiger Dateien zwischen zwei Rechnern, basierend auf dem UDP-Protokoll" zu erstellen. Als Programmiersprache war JAVA vorgegeben. Das Stop-and-Wait-Protokoll sollte implementiert werden.

2 Funktionsweise

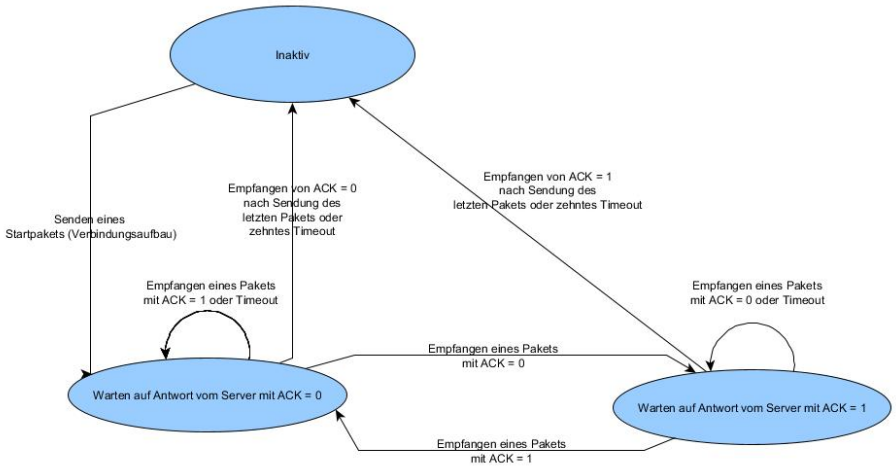
2.1 Client

Die Clientklasse erwartet drei Argumente: Host, PortNr, Datei. Zunächst wird geprüft, ob die Argumentenliste vollständig ist. Es wird geprüft, ob der Pfad/Dateiname, der als Parameter übergeben wurde, existiert. Andernfalls beendet sich der Client. Als Maximum Transmission Unit (MTU) sind 1500 Bytes gewählt. Vor der Übertragung wird die Anzahl der zu übertragenden Pakete berechnet, sowie die letzte und vorletzte Paketgröße. Dies ist notwendig, da alle anderen Pakete mit einer Anzahl von 1497 Informationsbytes versendet werden. Der CRC-Code soll im letzten Paket komplett versendet werden. Daher ist eine Größenberechnung des vorletzten Pakets notwendig. Der Client speichert die aktuelle Systemzeit um später Übertragungsinformationen mitteilen zu können. Der Client durchläuft eine Schleife bis alle Pakete erfolgreich übertragen werden.

Bis zur ersten erfolgreichen Übertragung wird das zu sendende Paket mit den Startpaket-Informationen(nach gegebenem Protokoll) gefüllt. Anschließend erfolgt die Übertragung der Datenpakete.

Am Ende der Schleife versendet der Client das Paket. Der Client startet eienen Timer. Sofern nach 1000ms (statisch gewähltes timeout) keine Antwort vom Server eintrifft, wird das Paket erneut versendet. Sollte die Antwort rechtzeitig eintreffen, wird geprüft, ob das erhaltene ACK mit dem versendeten ACK übereinstimmt. Sofern dies der Fall ist, wird der Erfolgszähler (Anzahl an erfolgreichen Paketübertragungen) um eins nach oben gezählt. Andernfalls versendet der Client dasselbe Paket erneut und setzt den Timeout-Timer zurück. Falls ein Timeout zehn mal hintereinander auftritt, wird das Programm beendet.

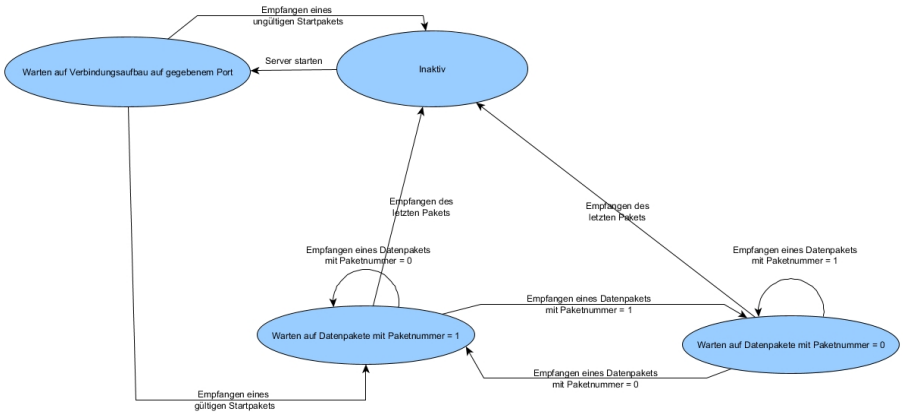
Figure 1: Zustandsdiagramm - Client



2.2 Server

Dem Server wird der Port, auf dem er lauschen soll, als Parameter übergeben. Er wartet auf die Ankuft eines Pakets. Sobald das erste Paket ankommt, prüft er, ob es die Kennung "Start" an festgelegter Position enthält. Ist dies nicht der Fall, beendet sich der Server. Es wird geprüft, ob der enthaltene CRC-Code über das Startpaket gültig sein kann. Nun liest der Server den erhaltenen Pfad/Dateinamen und modifiziert diesen, falls dieser im aktuellen Verzeichnis schon vorhanden sein sollte. Der Server liest nun die erhaltenen Datenpakete. Stimmt die erhaltene Paketnummer mit dem erwarteten ACKs überein, handelt es sich um ein neues Paket. Dessen Informationsbits werden ausgelesen und in die zu Beginn erstellte Datei geschrieben. Der Wert des erwarteten ACK wird um 1 (mod 2) erhöht. Gibt es keine Übereinstimmung, so sendet der Server ein Paket mit der zuletzt erhaltenen (validen) Paketnummer an den Client. Sobald alle Pakete übertragen wurden, überprüft der Server den erhaltenen CRC-Code über die Datei auf Gültigkeit. Anschließend wird dessen Ergebnis sowie einige Übertragungsinformationen ausgegeben und der Server beendet sich.

Figure 2: Zustandsdiagramm - Server



$$\alpha = \sqrt{\beta} \tag{1}$$

3 Protokollbewertung

3.1 Probleme

Ein Problem dieses Protokolls ist, dass das letzte ACK des Servers möglicherweise nicht angekommen ist, der Server dieses allerdings nicht erneut sendet, da er keine weiteren Pakete mehr erwartet. Somit kann der Client nicht feststellen, ob das letzte Paket angekommen ist.

3.2 Limitierungen

Die Datenrate bei diesem Protokoll verhält verhältnismäßig gering aus, da der Client mindestens eine ganze RTT (RTT * AnzahlVersucheBisErfolg) abwarten muss, bis er ein neues Paket lossenden kann. Außerdem wurde in diesem Fall keine Fehlerkorrektur implementiert. Fehlerhafte Pakete verursacht somit eine deutliche Verschlechterung der Datenrate.

3.3 Verbesserungsvorschläge

Besser wäre es, wenn die ACKs nicht binär wären und Der Client nicht auf die Antwort des Servers warten würde, sondern unmittelbar weitere Pakete losschickt in der Hoffung, dass die Übertragung fehlerfrei erfolgt. Im Fehlerfall springt der Client an jene Stelle zurück, an welcher der Server die Pakete als angekommen gemeldet hat. Denkbar wäre im Falle häufiger Paketverluste das doppelte Senden von identischen Paketen, sodass die Chance eines Verlustes halbiert wird.

4 Performance

max.erzielbaren Durchsatz bei 10% Paketverlust und 10 ms Verzögerung für ein einzelnes Paket:

$$\eta_{sw} = \frac{T_p}{T_p + T_\omega}(1 - P_{de})(1 - P_{rü})R \tag{2}$$

$$geg : T_a = 10ms, P_{de} = P_{rü} = 0.1, R = 1497/1500 = 0,998 \tag{3}$$

Verbindungsgeschwindigkeit in meinem Lan: 135 MB/ S

$$T_p = \frac{150B * s}{135000000B} = 0.000001s = 0,001ms \tag{4}$$

$$T_{ACK} = 3B * S/135000000B = 0,0002ms \tag{5}$$

$$T_\omega = 2 * T_a + T_{ACK} \approx 20ms, \tag{6}$$

$$\eta_{sw} = \frac{0,001}{0,001 + 20}(0.9)(0.9)0.998 = 0,00004 = 0,004\% \tag{7}$$

Der niedrige Durchsatz kommt dadurch zustande, dass nur eine sehr geringe Menge an Bytes gesendet wird und auf dessen Antwort gewartet werden muss. In der Praxis wird die Übertragungsverzögerung niedriger sein und zwar aufgrund der Verwaltungskosten, also Startpaket, Validitätsprüfung, Speicherzugriffe..

Ergebnis für ein in der Praxis zufällig gewähltes einziges Paket:

$$T_\omega + T_P = 39ms, Annahme : T_\omega = 20ms => \eta_{sw} = 39,4\% \tag{8}$$